BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

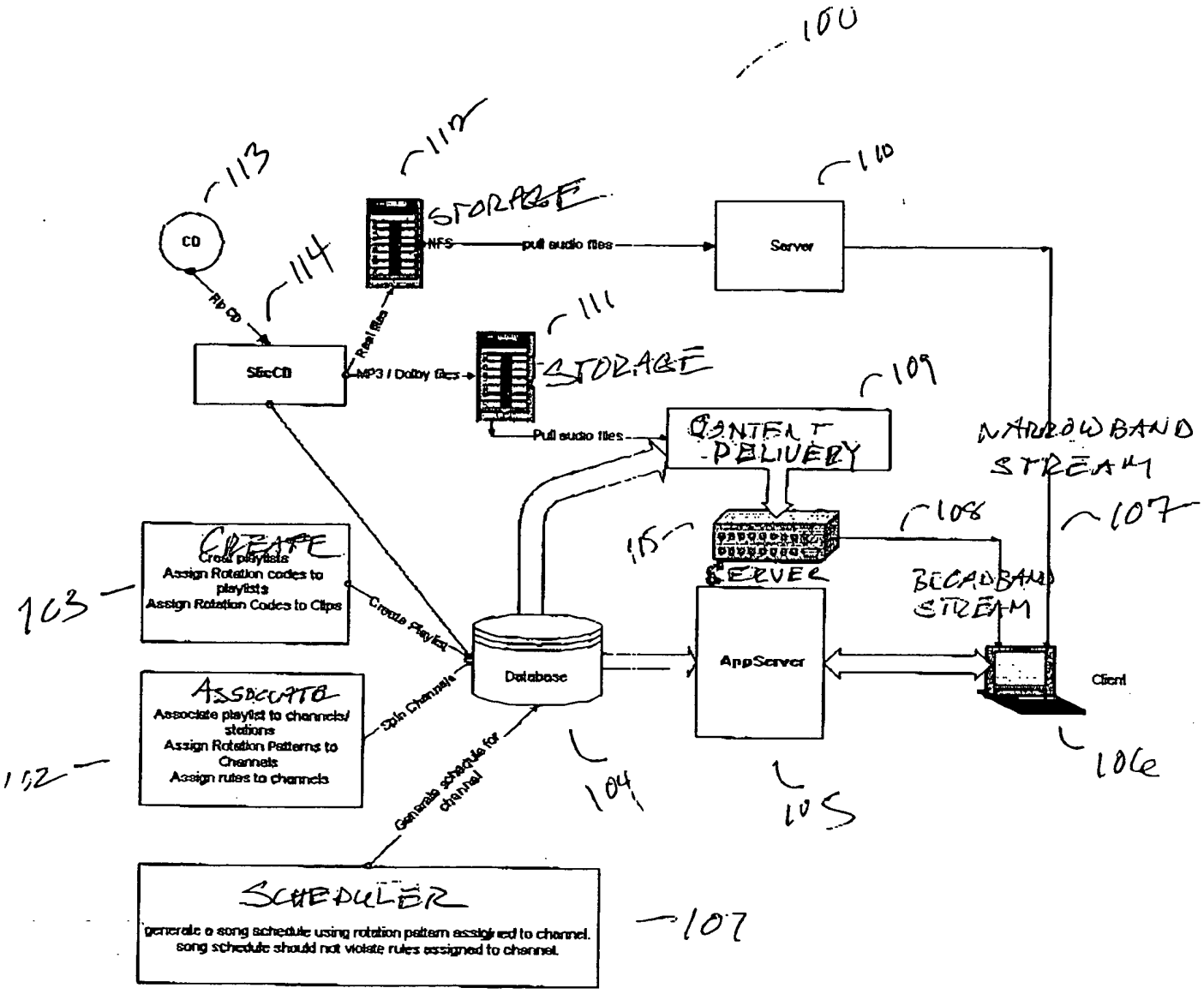
100
113
CD
Rip CD
114
SecCD
111
STORAGE
Real files
MP3 / Dolby files
NFS
pull audio files
112
STORAGE
Server
110
Pull audio files
CONTENT DELIVERY
109
NARROWBAND STREAM
107
115
SERVER
108
BROADBAND STREAM
CREATE
Create playlists
Assign Rotation codes to playlists
Assign Rotation Codes to Clips
103
Create playlists
ASSOCIATE
Associate playlist to channels/ stations
Assign Rotation Patterns to Channels
Assign rules to channels
102
Spin Channels
Generate schedule for channel
Database
AppServer
Client
104
105
106
SCHEDULER
generate a song schedule using rotation pattern assigned to channel. song schedule should not violate rules assigned to channel.
107

Figure 1

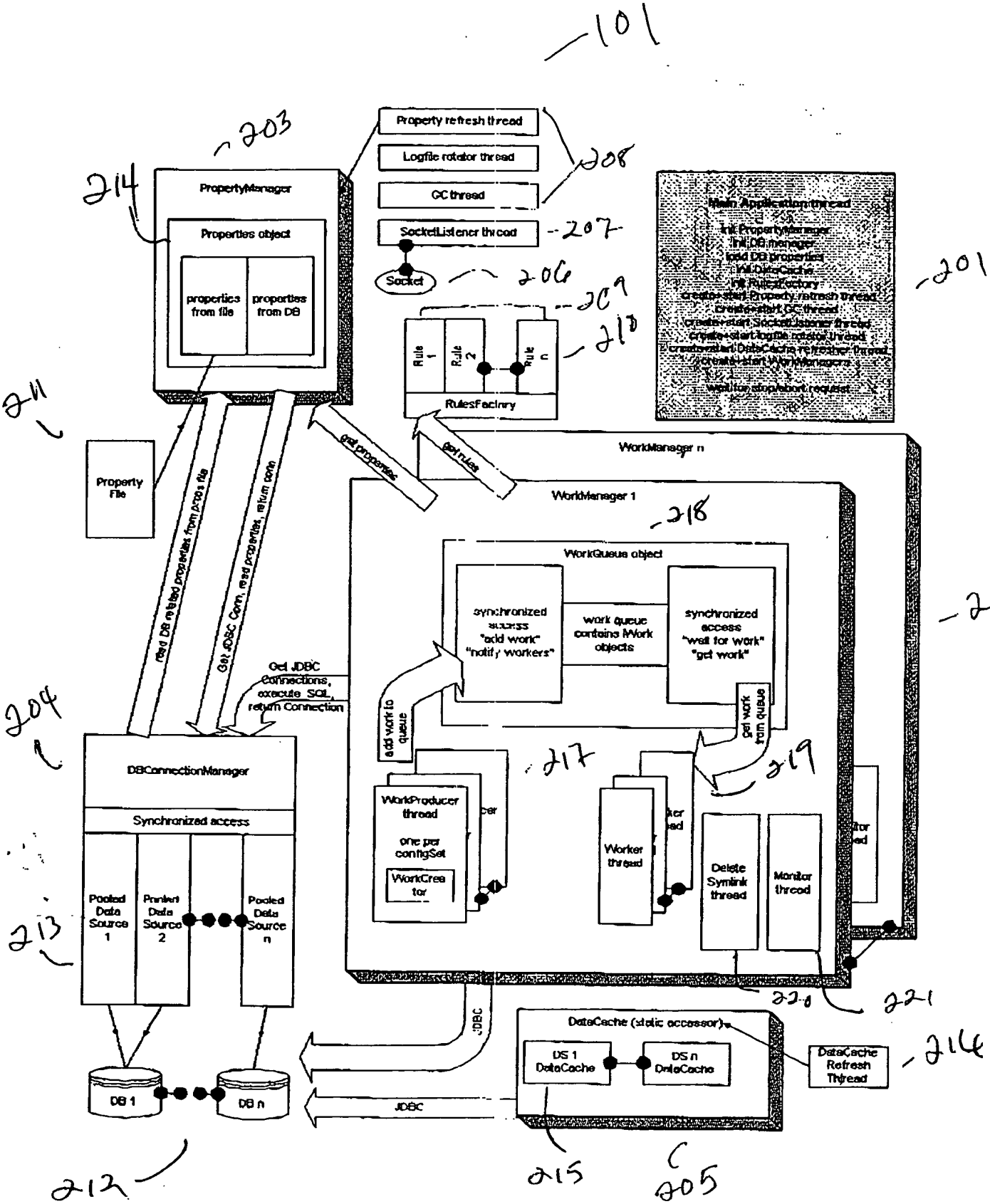
101
203
214
PropertyManager
Properties object
properties from file
properties from DB
Property refresh thread
Logfile rotator thread
GC thread
SocketListener thread
208
207
Socket
206
209
210
Rule 1
Rule 2
Rule n
RulesFactory
201
Main Application thread
Init PropertyManager
Init DB manager
load DB properties
Init DataCache
Init RulesFactory
create+start Property refresh thread
create+start GC thread
create+start SocketListener thread
create+start logfile rotator thread
create+start DataCache refresh thread
create+start WorkManagers
wait for stop/abort request
211
Property file
read DB related properties from ppds file
Get JDBC Conn, read properties, return conn
get properties
get rules
WorkManager n
WorkManager 1
218
WorkQueue object
synchronized access "add work" "notify workers"
work queue contains IWork objects
synchronized access "wait for work" "get work"
202
Get JDBC Connections, execute SQL, return Connection
add work to queue
get work from queue
204
DBConnectionManager
Synchronized access
217
219
WorkProducer thread
one per configSet
WorkCreator
Worker thread
Delete Symlink thread
Monitor thread
Pooled Data Source 1
Pooled Data Source 2
Pooled Data Source n
213
220
221
JDBC
DataCache (static accessor)
DS 1 DataCache
DS n DataCache
DataCache Refresh Thread
216
DB 1
DB n
JDBC
212
215
205

Figure 2

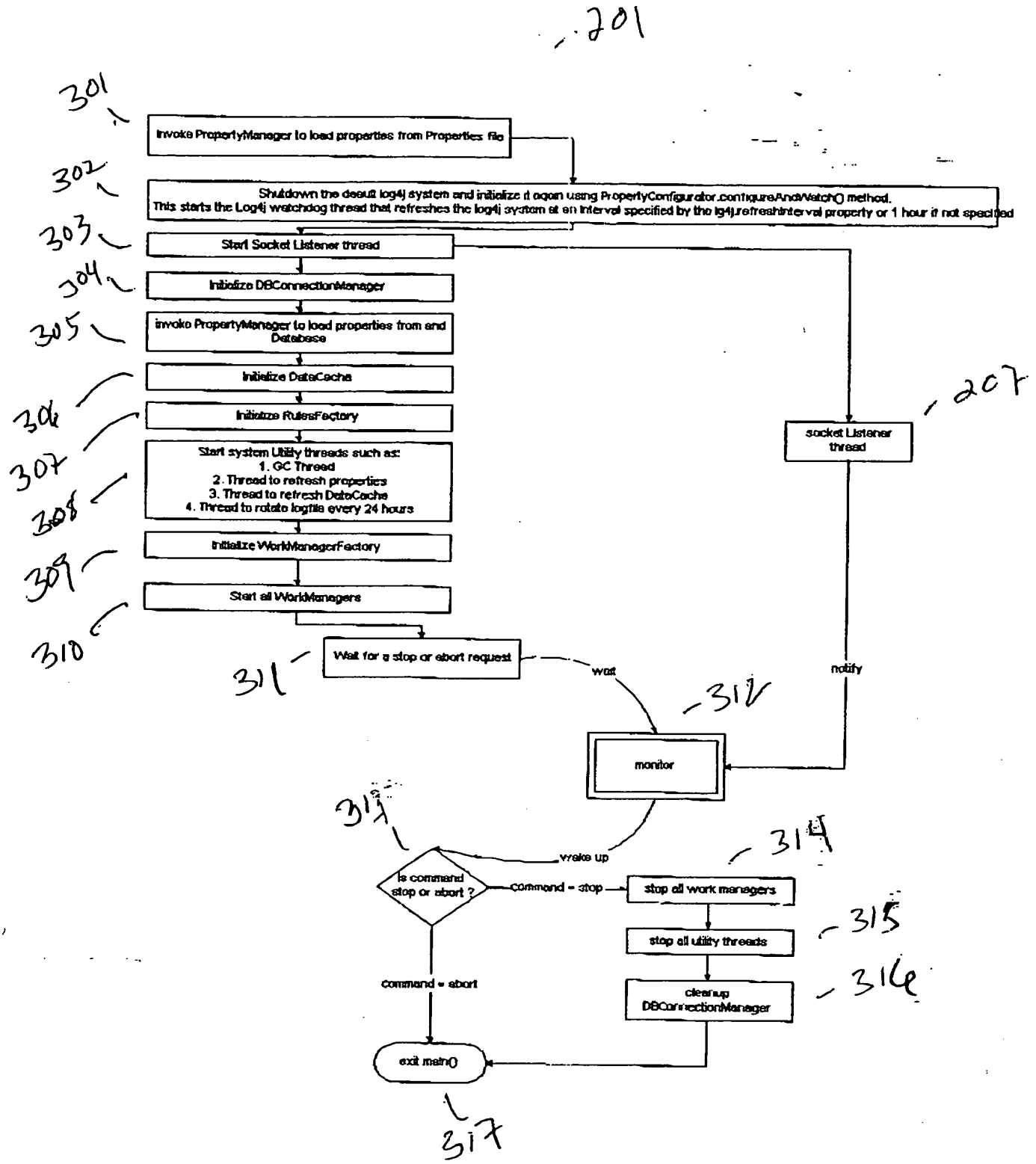
301
Invoke PropertyManager to load properties from Properties file
302
Shutdown the deault log4j system and initialize it again using PropertyConfigurator.configureAndWatch() method.
This starts the Log4j watchdog thread that refreshes the log4j system at an interval specified by the lg4j.refreshInterval property or 1 hour if not specified
303
Start Socket Listener thread
304
Initialize DBConnectionManager
305
Invoke PropertyManager to load properties from and Database
306
Initialize DataCache
307
Initialize RulesFactory
308
Start system Utility threads such as:
1. GC Thread
2. Thread to refresh properties
3. Thread to refresh DataCache
4. Thread to rotate logfile every 24 hours
309
Initialize WorkManagerFactory
310
Start all WorkManagers
311
Wait for a stop or abort request
wait
312
monitor
notify
207
socket Listener thread
313
Is command stop or abort ?
wake up
command = stop
314
stop all work managers
315
stop all utility threads
316
cleanup DBConnectionManager
command = abort
exit main()
317

Figure 3

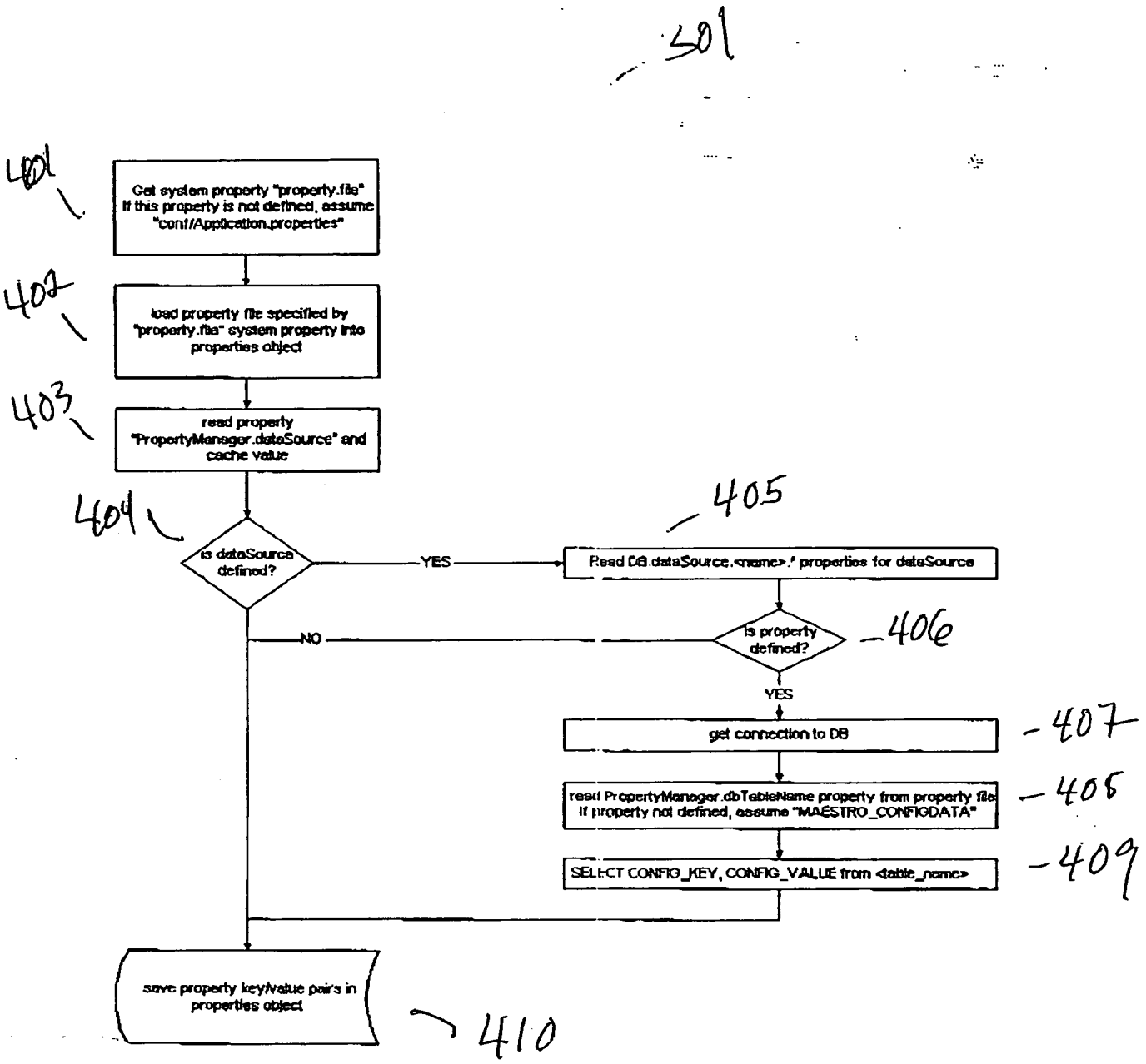
401
Get system property "property.file"
If this property is not defined, assume
"conf/Application.properties"
402
load property file specified by
"property.file" system property into
properties object
403
read property
"PropertyManager.dataSource" and
cache value
404
is dataSource
defined?
YES
405
Read DB.dataSource.<name>.* properties for dataSource
NO
is property
defined?
406
YES
get connection to DB
407
read PropertyManager.dbTableName property from property file
If property not defined, assume "MAESTRO_CONFIGDATA"
408
SELECT CONFIG_KEY, CONFIG_VALUE from <table_name>
409
save property key/value pairs in
properties object
410

Figure 4

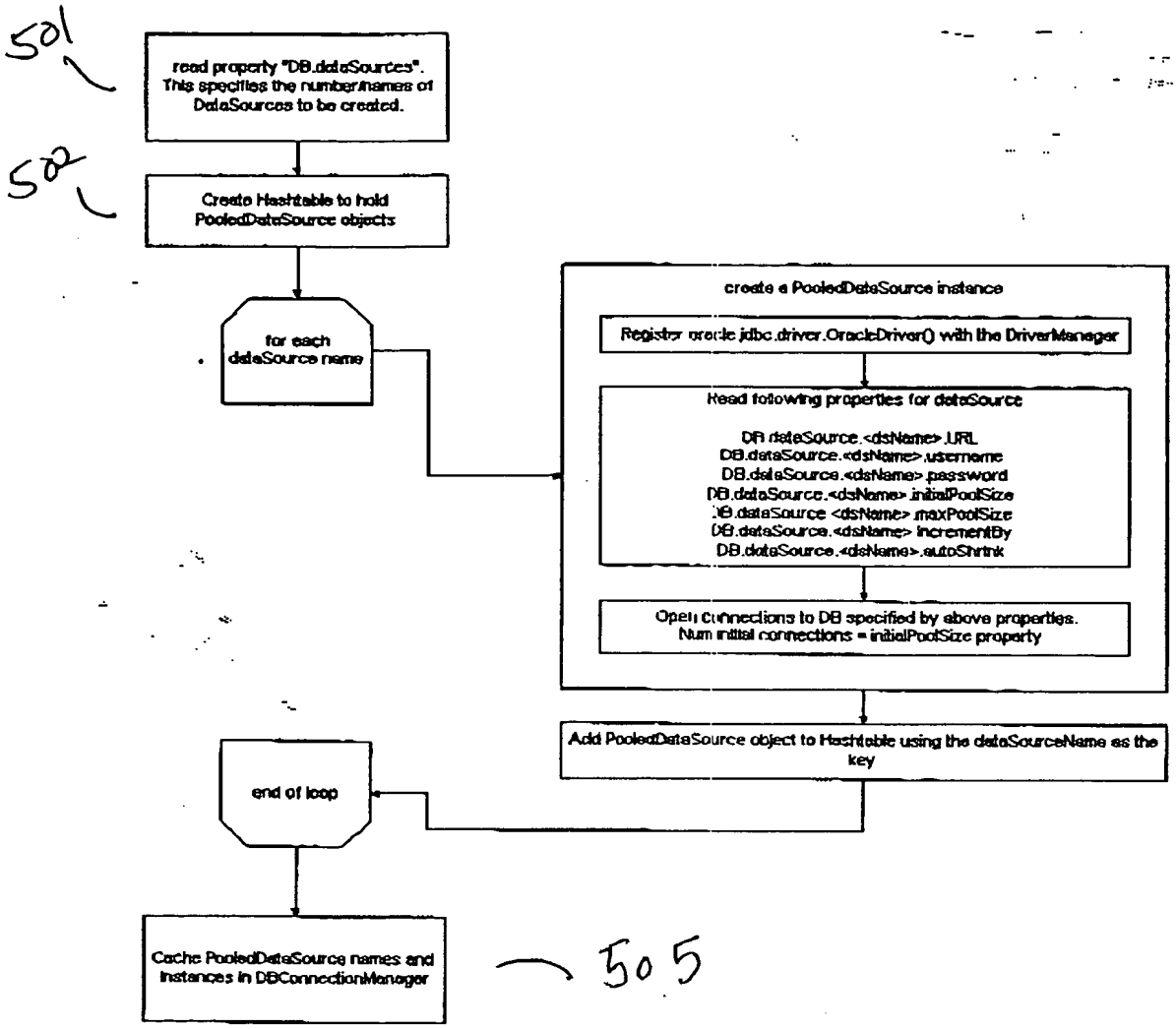
304
501
read property "DB.dataSources". This specifies the number/names of DataSources to be created.
502
Create Hashtable to hold PooledDataSource objects
for each dataSource name
create a PooledDataSource instance
Register oracle.jdbc.driver.OracleDriver() with the DriverManager
Read following properties for dataSource
DB.dataSource.<dsName>.URL
DB.dataSource.<dsName>.username
DB.dataSource.<dsName>.password
DB.dataSource.<dsName>.initialPoolSize
DB.dataSource.<dsName>.maxPoolSize
DB.dataSource.<dsName>.incrementBy
DB.dataSource.<dsName>.autoShrink
503
Open connections to DB specified by above properties. Num initial connections = initialPoolSize property
504
Add PooledDataSource object to Hashtable using the dataSourceName as the key
end of loop
Cache PooledDataSource names and instances in DBConnectionManager
505

Figure 5

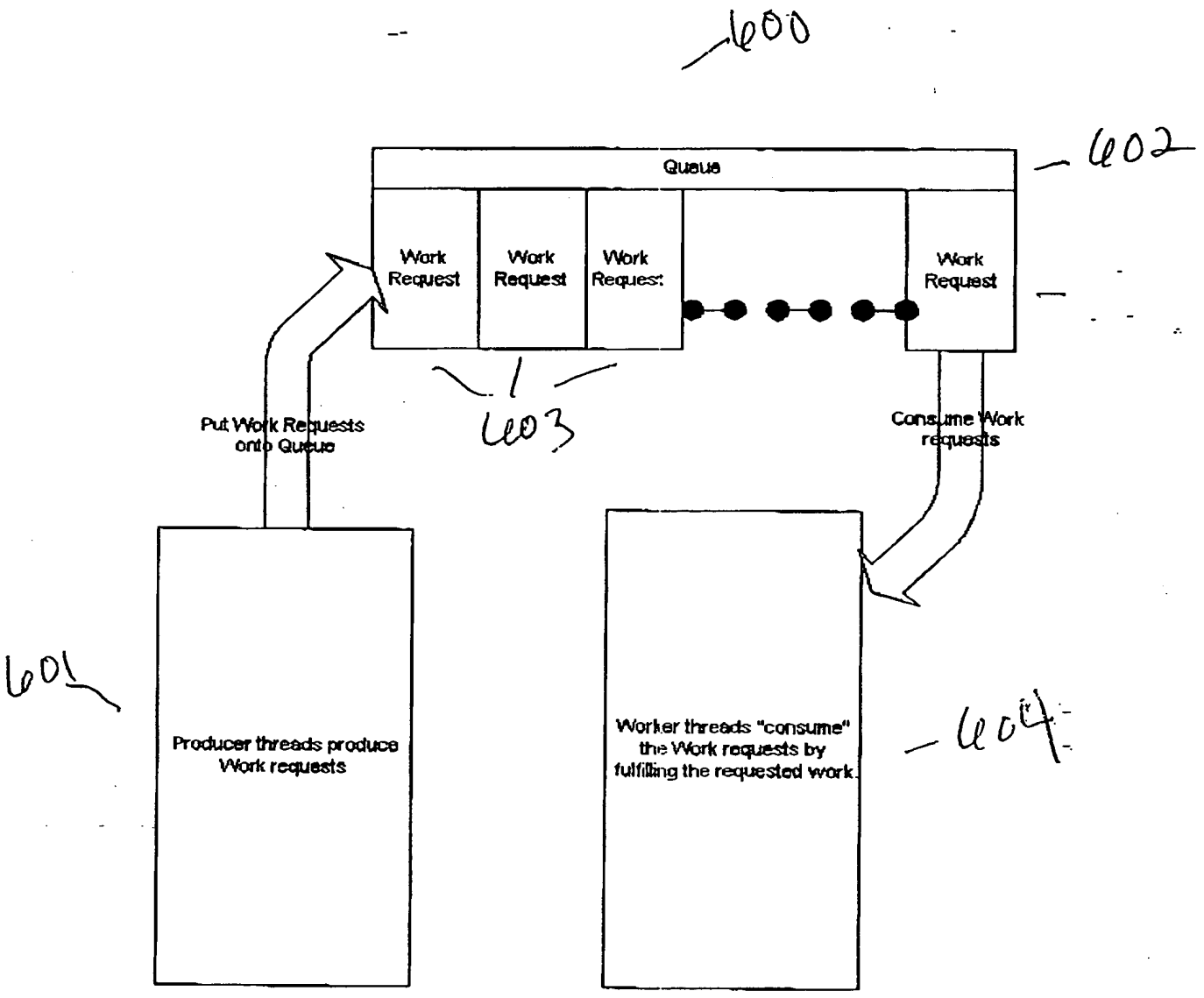
600
Queue
602
Work Request
Work Request
Work Request
Work Request
603
Put Work Requests onto Queue
Consume Work requests
601
Producer threads produce Work requests
Worker threads "consume" the Work requests by fulfilling the requested work.
604

Figure 6

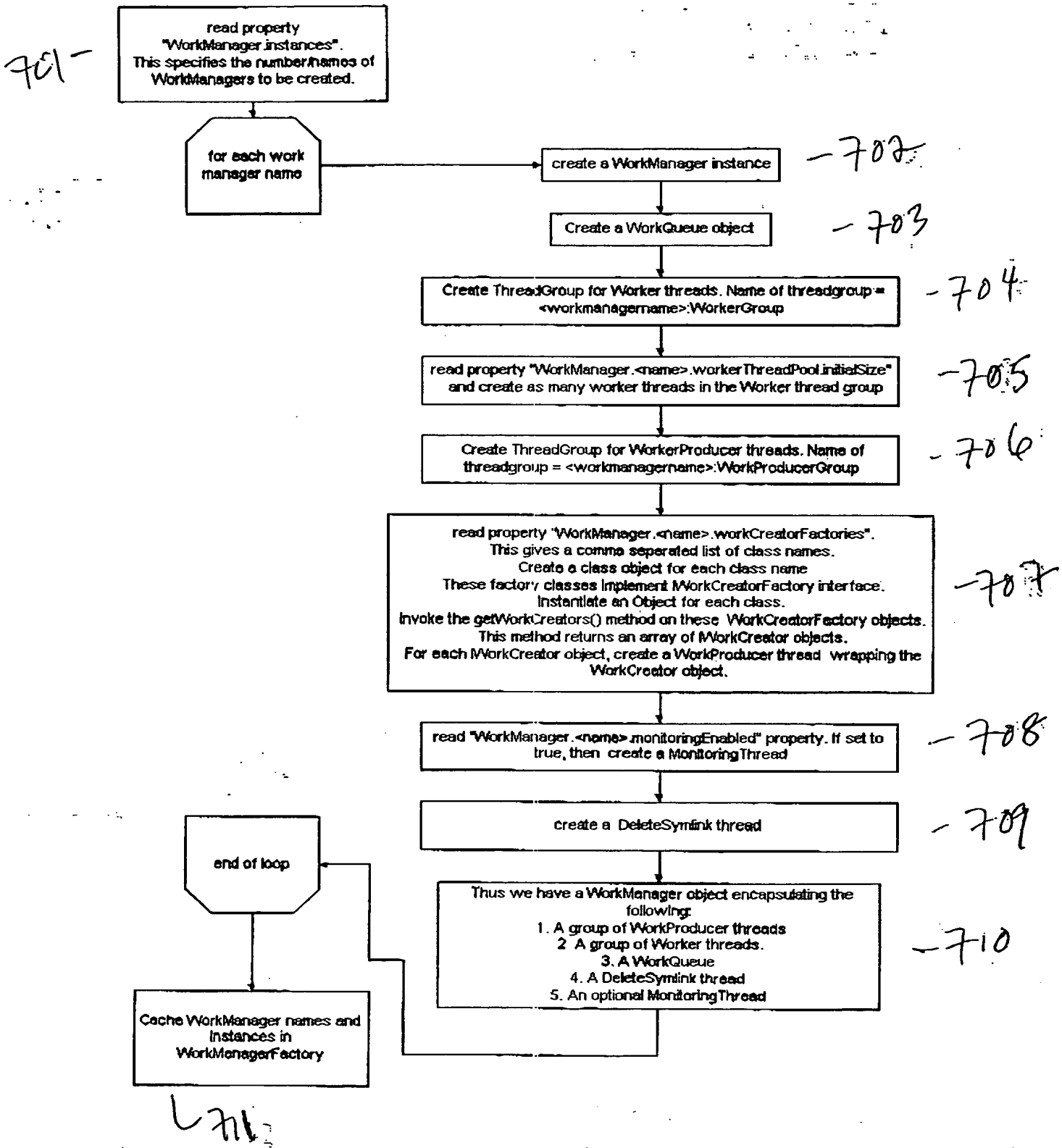
read property "WorkManager.instances". This specifies the number/names of WorkManagers to be created.
for each work manager name
create a WorkManager instance
Create a WorkQueue object
Create ThreadGroup for Worker threads. Name of threadgroup = <workmanagername>:WorkerGroup
read property "WorkManager.<name>.workerThreadPool.initialSize" and create as many worker threads in the Worker thread group
Create ThreadGroup for WorkerProducer threads. Name of threadgroup = <workmanagername>:WorkProducerGroup
read property "WorkManager.<name>.workCreatorFactories". This gives a comma seperated list of class names. Create a class object for each class name These factory classes implement IWorkCreatorFactory interface. Instantiate an Object for each class. Invoke the getWorkCreators() method on these WorkCreatorFactory objects. This method returns an array of IWorkCreator objects. For each IWorkCreator object, create a WorkProducer thread wrapping the WorkCreator object.
read "WorkManager.<name>.monitoringEnabled" property. If set to true, then create a MonitoringThread
create a DeleteSymlink thread
Thus we have a WorkManager object encapsulating the following: 1. A group of WorkProducer threads 2 A group of Worker threads. 3. A WorkQueue 4. A DeleteSymlink thread 5. An optional MonitoringThread
end of loop
Cache WorkManager names and instances in WorkManagerFactory
701
707
703
704
705
706
707
708
709
710
711

Figure 7

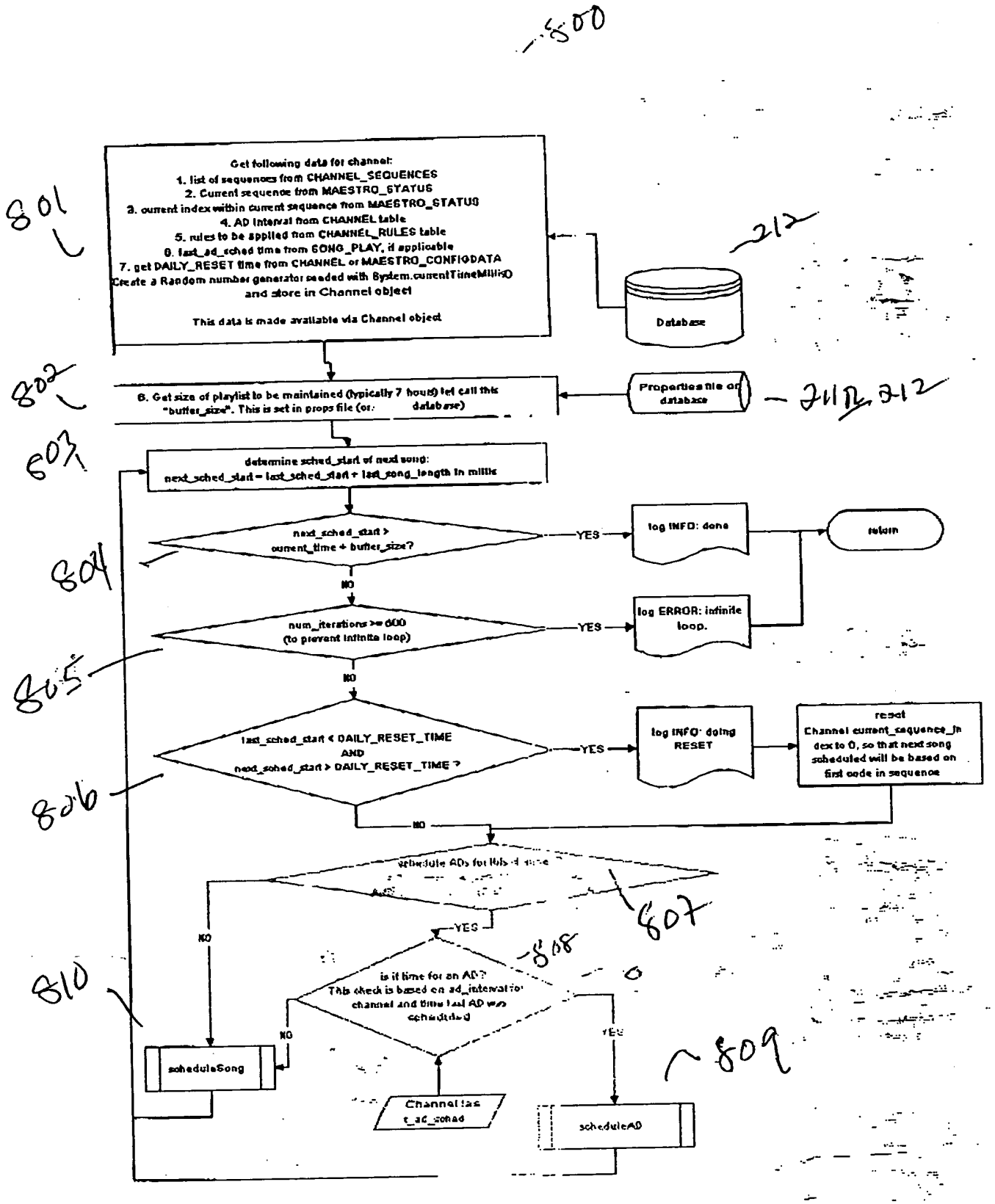
800
Get following data for channel:
1. list of sequences from CHANNEL_SEQUENCES
2. Current sequence from MAESTRO_STATUS
3. current index within current sequence from MAESTRO_STATUS
4. AD interval from CHANNEL table
5. rules to be applied from CHANNEL_RULES table
6. last_ad_sched time from SONG_PLAY, if applicable
7. get DAILY_RESET time from CHANNEL or MAESTRO_CONFIGDATA
Create a Random number generator seeded with System.currentTimeMillis()
and store in Channel object
This data is made available via Channel object
801
212
Database
802
6. Get size of playlist to be maintained (typically 7 hours) let call this "buffer_size". This is set in props file (or: database)
Properties file or database
211 & 212
803
determine sched_start of next song:
next_sched_start = last_sched_start + last_song_length in millis
804
next_sched_start > current_time + buffer_size?
YES
log INFO: done
return
NO
805
num_iterations >= 600 (to prevent infinite loop)
YES
log ERROR: infinite loop.
NO
806
last_sched_start < DAILY_RESET_TIME AND next_sched_start > DAILY_RESET_TIME ?
YES
log INFO: doing RESET
reset Channel current_sequence_index to 0, so that next song scheduled will be based on first code in sequence
NO
schedule ADs for this time
807
YES
NO
808
is it time for an AD? This check is based on ad_interval for channel and time last AD was scheduled
NO
YES
809
scheduleSong
Channel.las t_ad_sched
scheduleAD
810

Figure 8

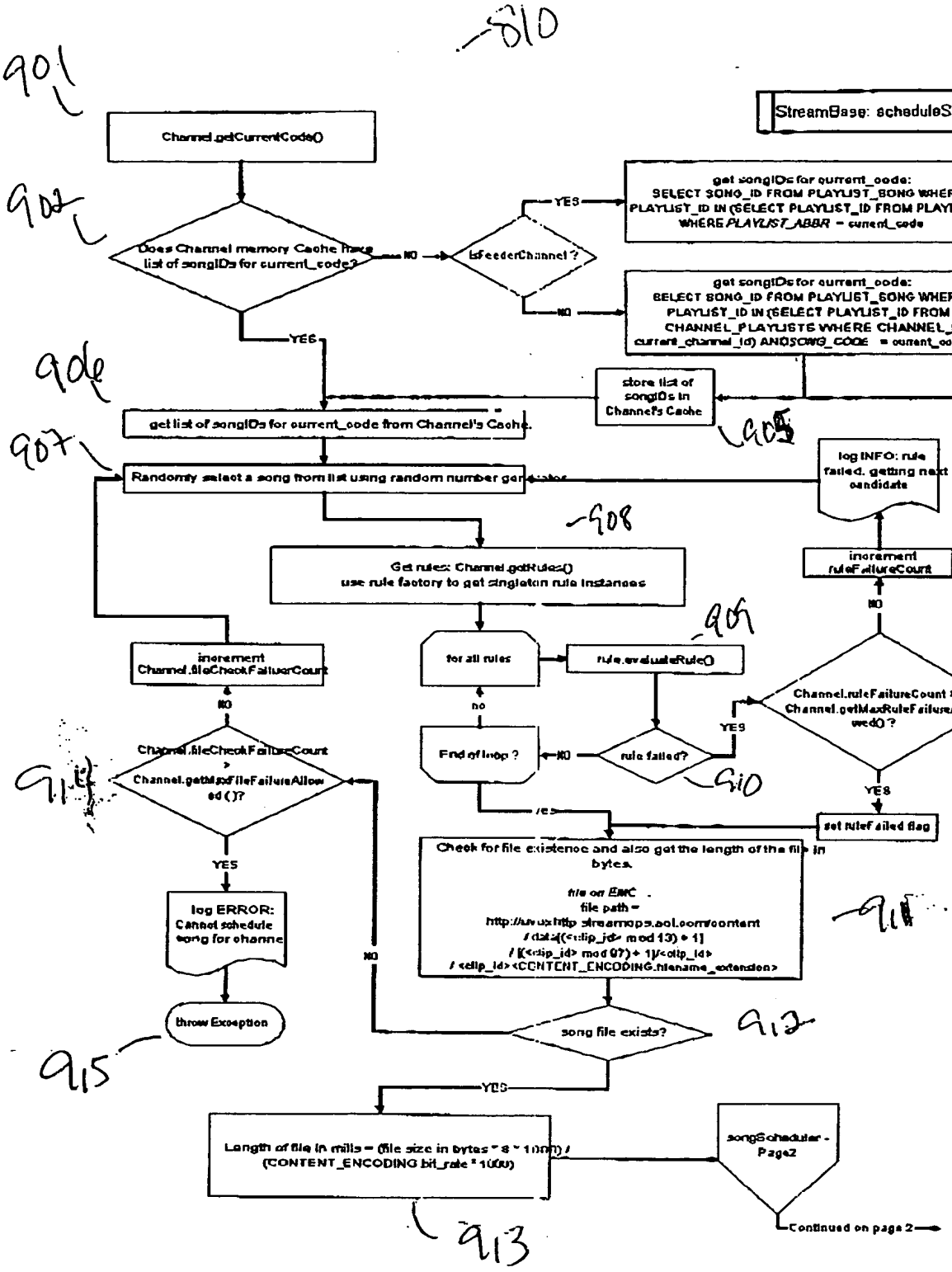
910
StreamBase: scheduleSong
901
Channel.getCurrentCode()
902
Does Channel memory Cache have list of songIDs for current_code?
NO
isFeederChannel ?
YES
NO
YES
903
get songIDs for current_code: SELECT SONG_ID FROM PLAYLIST_SONG WHERE PLAYLIST_ID IN (SELECT PLAYLIST_ID FROM PLAYLISTS WHERE PLAYLIST_ABBR = current_code
904
get songIDs for current_code: SELECT SONG_ID FROM PLAYLIST_SONG WHERE PLAYLIST_ID IN (SELECT PLAYLIST_ID FROM CHANNEL_PLAYLISTS WHERE CHANNEL_ID = current_channel_id) AND SONG_CODE = current_code
905
store list of songIDs in Channel's Cache
906
get list of songIDs for current_code from Channel's Cache.
907
Randomly select a song from list using random number generator
908
Get rules: Channel.getRules()
use rule factory to get singleton rule instances
909
for all rules
rule.evaluateRule()
910
rule failed?
NO
End of loop ?
no
YES
911
Channel.fileCheckFailureCount > Channel.getMaxFileFailureAllowed()?
NO
YES
912
Check for file existence and also get the length of the file in bytes.
file on EMC
file path =
http://xxx.xxx.http streamops.aol.com/content
/ data/((<clip_id> mod 13) + 1)
/ ((<clip_id> mod 97) + 1)/<clip_id>
/ <clip_id><CONTENT_ENCODING.filename_extension>
song file exists?
NO
YES
913
Length of file in mills = (file size in bytes * 8 * 1000) / (CONTENT_ENCODING.bit_rate * 1000)
914
Channel.ruleFailureCount >= Channel.getMaxRuleFailureAllowed() ?
NO
YES
915
log ERROR: Cannot schedule song for channe
throw Exception
916
set ruleFailed flag
917
increment Channel.fileCheckFailureCount
918
increment ruleFailureCount
919
log INFO: rule failed. getting next candidate
songScheduler - Page2
Continued on page 2

Figure 9

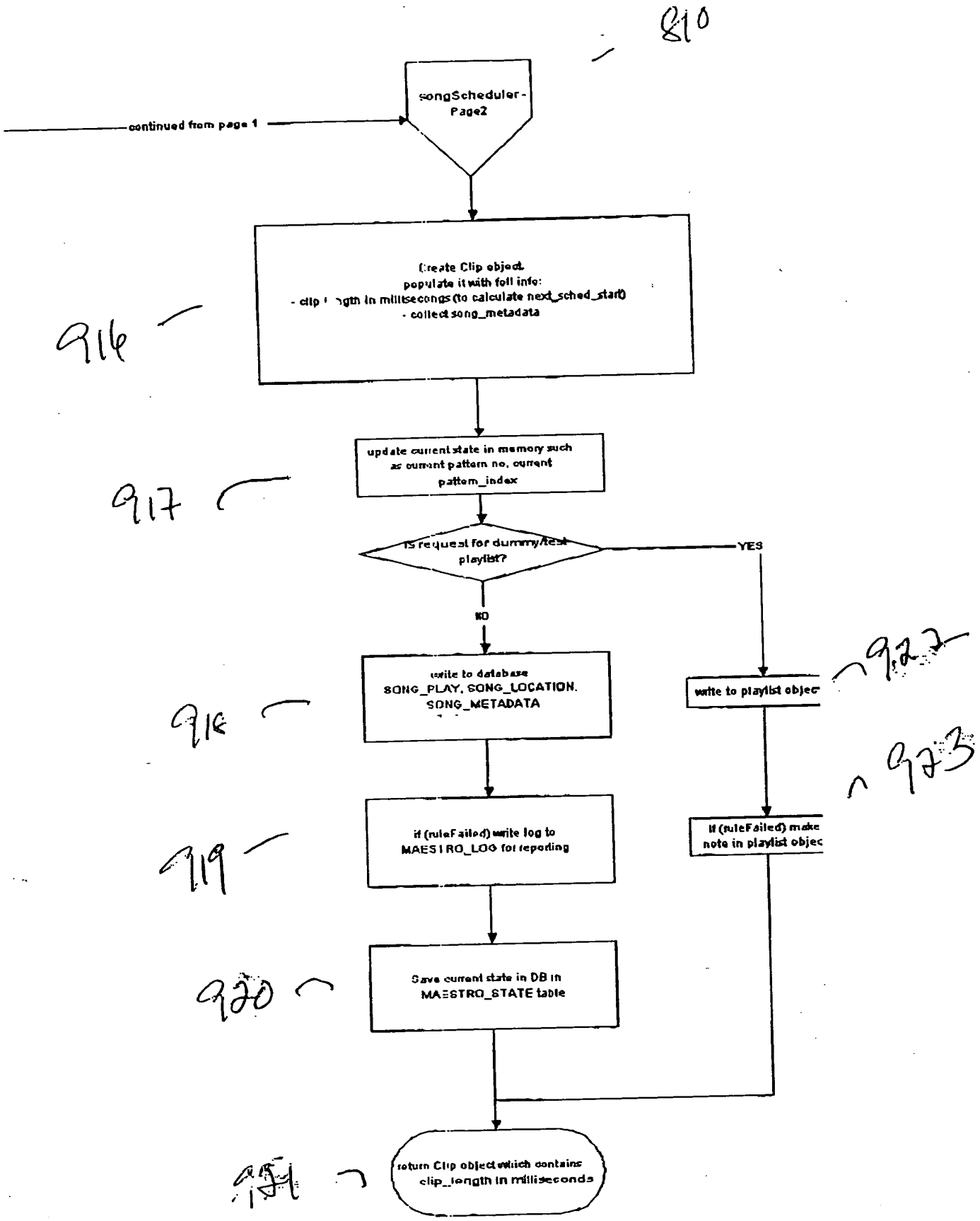
810
songScheduler - Page2
continued from page 1
Create Clip object
populate it with foll info:
- clip length in milliseconds (to calculate next_sched_start)
- collect song_metadata
916
update current state in memory such as current pattern no, current pattern_index
917
Is request for dummy/test playlist?
YES
NO
write to database SONG_PLAY, SONG_LOCATION, SONG_METADATA
916
write to playlist objec
922
923
if (ruleFailed) write log to MAESTRO_LOG for reporting
919
if (ruleFailed) make note in playlist objec
Save current state in DB in MAESTRO_STATE table
920
return Clip object which contains clip_length in milliseconds
921

Figure 10

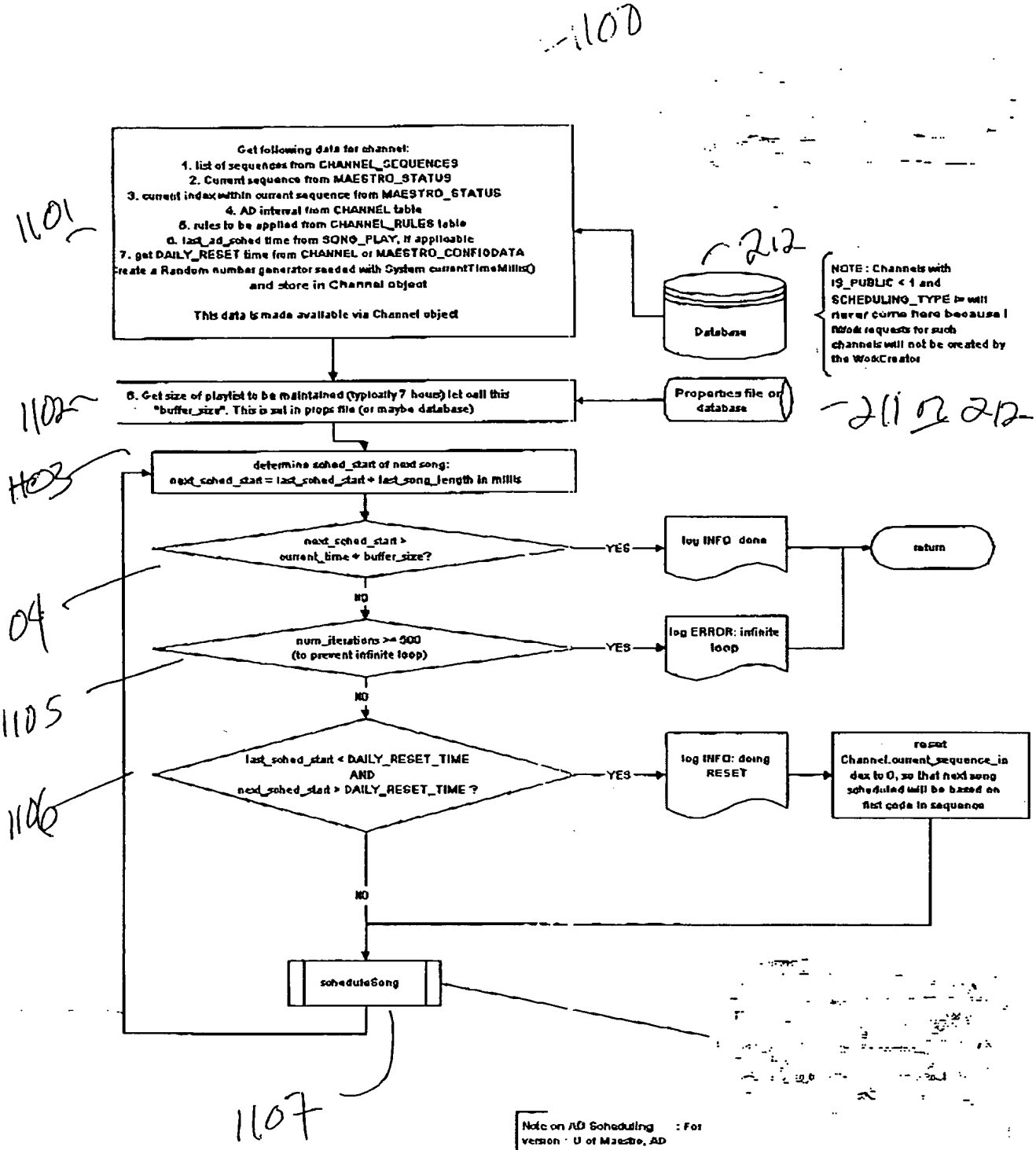
Get following data for channel:
1. list of sequences from CHANNEL_SEQUENCES
2. Current sequence from MAESTRO_STATUS
3. current index within current sequence from MAESTRO_STATUS
4. AD interval from CHANNEL table
5. rules to be applied from CHANNEL_RULES table
6. last_ad_sched time from SONG_PLAY, if applicable
7. get DAILY_RESET time from CHANNEL or MAESTRO_CONFIGDATA
Create a Random number generator seeded with System currentTimeMillis()
and store in Channel object
This data is made available via Channel object
Database
NOTE : Channels with
IS_PUBLIC < 1 and
SCHEDULING_TYPE != will
never come here because I
think requests for such
channels will not be created by
the WorkCreator
6. Get size of playlist to be maintained (typically 7 hours) let call this
"buffer_size". This is set in props file (or maybe database)
Properties file or
database
determine sched_start of next song:
next_sched_start = last_sched_start + last_song_length in millis
next_sched_start >
current_time + buffer_size?
YES
log INFO done
return
NO
num_iterations >= 500
(to prevent infinite loop)
YES
log ERROR: infinite
loop
NO
last_sched_start < DAILY_RESET_TIME
AND
next_sched_start > DAILY_RESET_TIME ?
YES
log INFO: doing
RESET
reset
Channel.current_sequence_in
dex to 0, so that next song
scheduled will be based on
first cuts in sequence
NO
scheduleSong
Note on AD Scheduling : For
version 1.0 of Maestro, AD
scheduling will be done by a
seperate thread in the system,
not the SongScheduler
When we integrate with
LightningCast, AD scheduling
will be done as part of the
SongScheduling logic

Figure 11

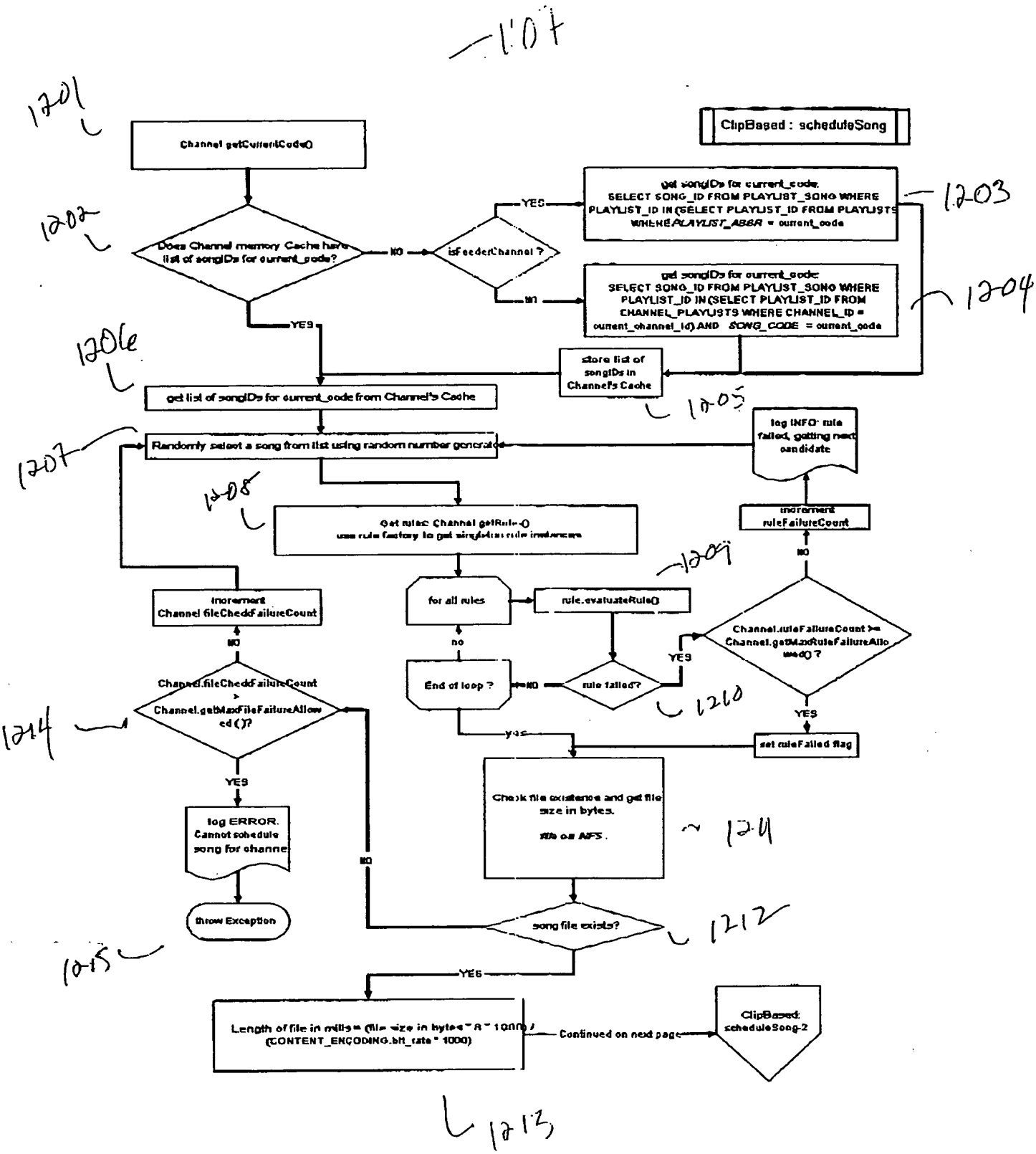
ClipBased : scheduleSong
Channel getCurrentCode()
Does Channel memory Cache have list of songIDs for current_code?
NO
isFeederChannel ?
YES
NO
get songIDs for current_code: SELECT SONG_ID FROM PLAYLIST_SONG WHERE PLAYLIST_ID IN (SELECT PLAYLIST_ID FROM PLAYLISTS WHERE PLAYLIST_ABBR = current_code
get songIDs for current_code: SELECT SONG_ID FROM PLAYLIST_SONG WHERE PLAYLIST_ID IN (SELECT PLAYLIST_ID FROM CHANNEL_PLAYLISTS WHERE CHANNEL_ID = current_channel_id) AND SONG_CODE = current_code
store list of songIDs in Channel's Cache
YES
get list of songIDs for current_code from Channel's Cache
Randomly select a song from list using random number generator
log INFO: rule failed, getting next candidate
Get rules: Channel.getRules()
use rule factory to get singleton rule instances
increment ruleFailureCount
NO
for all rules
rule.evaluateRule()
no
End of loop ?
NO
rule failed?
YES
Channel.ruleFailureCount >= Channel.getMaxRuleFailureAllowed() ?
YES
set ruleFailed flag
yes
increment Channel.fileCheckFailureCount
NO
Channel.fileCheckFailureCount > Channel.getMaxFileFailureAllowed()?
YES
log ERROR: Cannot schedule song for channe
throw Exception
Check file existence and get file size in bytes.
via an AFS.
song file exists?
NO
YES
Length of file in millis = (file size in bytes * 8 * 1000) / (CONTENT_ENCODING.bit_rate * 1000)
Continued on next page
ClipBased: scheduleSong-2
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1212
1213
1214
1215

Figure 12

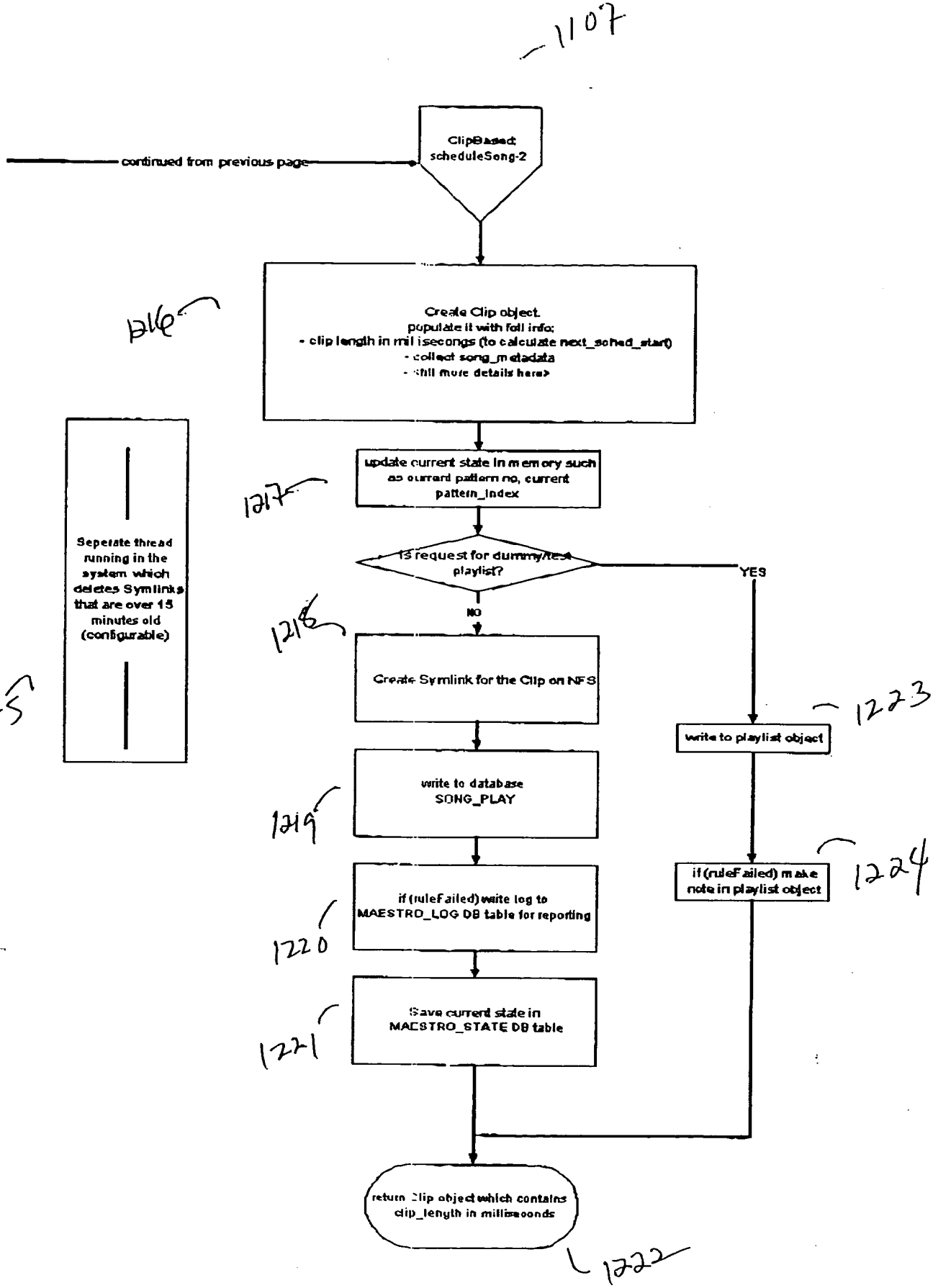
1107
continued from previous page
ClipBased scheduleSong-2
1216
Create Clip object.
populate it with foll info:
- clip length in milliseconds (to calculate next_sched_start)
- collect song_metadata
- <fill more details here>
1217
update current state in memory such as current pattern no, current pattern_index
Is request for dummy/test playlist?
YES
NO
1218
Create Symlink for the Clip on NFS
1219
write to database SONG_PLAY
1220
if (ruleFailed) write log to MAESTRO_LOG DB table for reporting
1221
Save current state in MAESTRO_STATE DB table
1223
write to playlist object
1224
if (ruleFailed) make note in playlist object
return Clip object which contains clip_length in milliseconds
1222
Seperate thread running in the system which deletes Symlinks that are over 15 minutes old (configurable)
1225

Figure 13

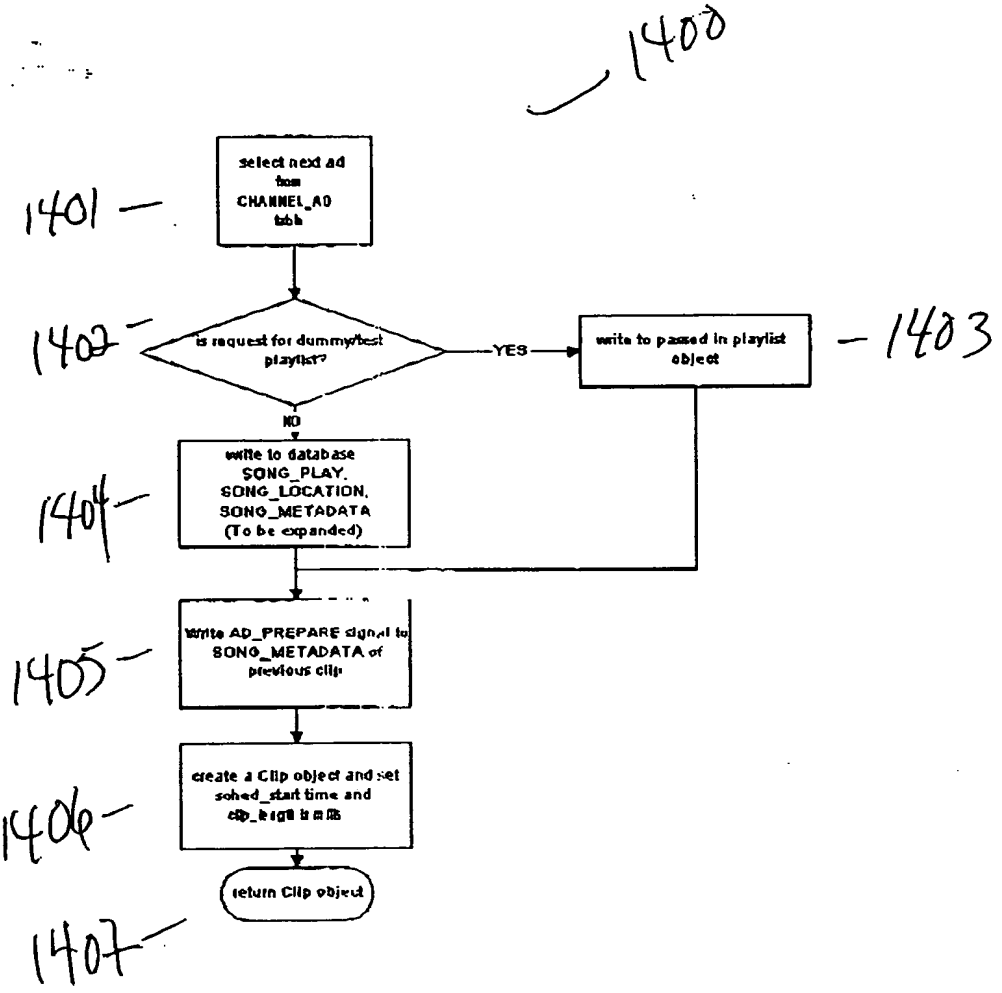
1400
select next ad from CHANNEL_AD table
1401
is request for dummy/best playlist?
1402
YES
write to passed in playlist object
1403
NO
write to database SONG_PLAY, SONG_LOCATION, SONG_METADATA (To be expanded)
1404
Write AD_PREPARE signal to SONG_METADATA of previous clip
1405
create a Clip object and set sched_start time and clip_length in ms
1406
return Clip object
1407

Figure 14

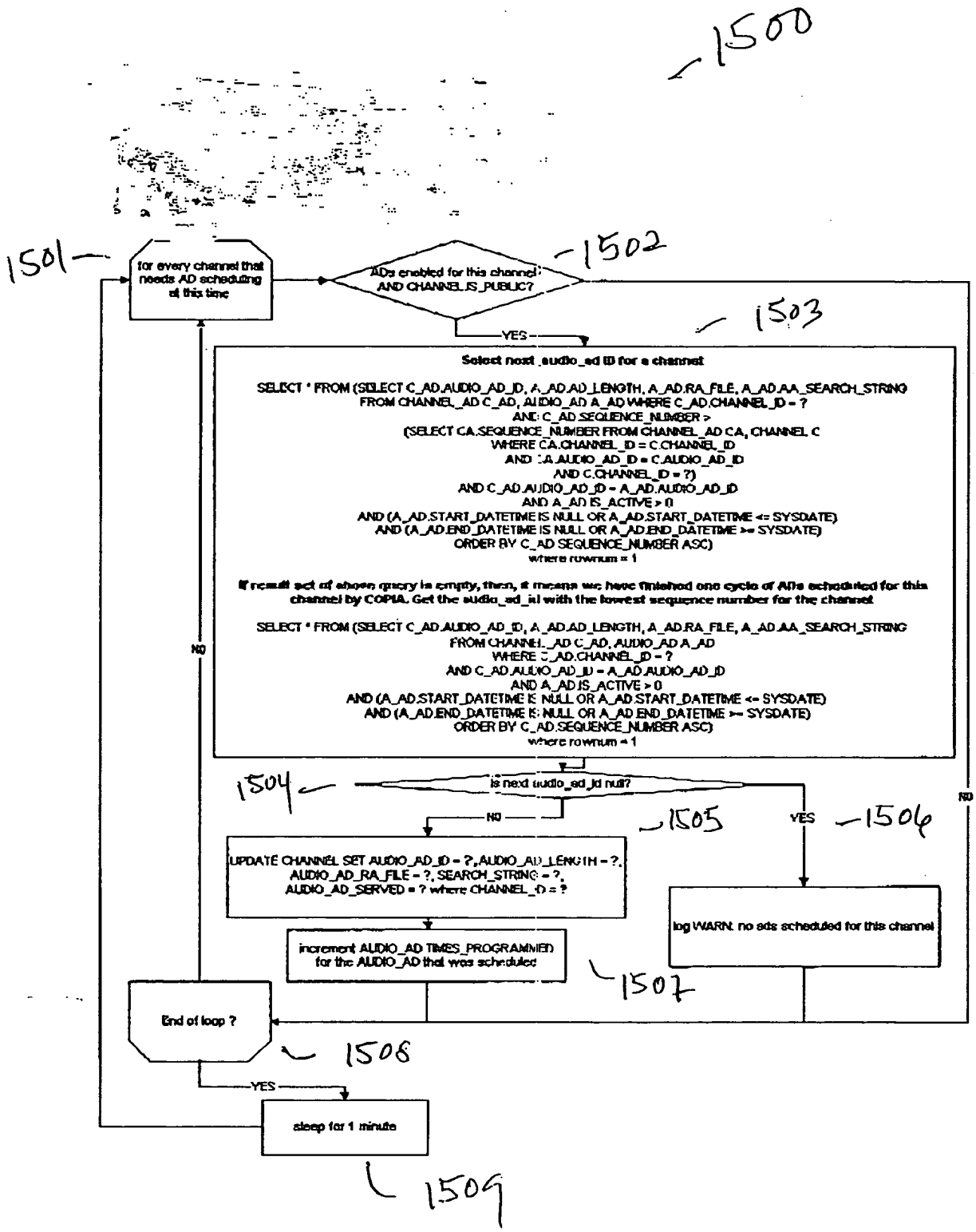
1500
1501
for every channel that needs AD scheduling at this time
1502
ADs enabled for this channel? AND CHANNEL IS_PUBLIC?
YES
1503
Select next audio_ad ID for a channel
SELECT * FROM (SELECT C_AD.AUDIO_AD_ID, A_AD.AD_LENGTH, A_AD.RA_FILE, A_AD.AA_SEARCH_STRING
FROM CHANNEL_AD C_AD, AUDIO_AD A_AD WHERE C_AD.CHANNEL_ID = ?
AND C_AD.SEQUENCE_NUMBER >
(SELECT CA.SEQUENCE_NUMBER FROM CHANNEL_AD CA, CHANNEL C
WHERE CA.CHANNEL_ID = C.CHANNEL_ID
AND CA.AUDIO_AD_ID = C.AUDIO_AD_ID
AND C.CHANNEL_ID = ?)
AND C_AD.AUDIO_AD_ID = A_AD.AUDIO_AD_ID
AND A_AD.IS_ACTIVE > 0
AND (A_AD.START_DATETIME IS NULL OR A_AD.START_DATETIME <= SYSDATE)
AND (A_AD.END_DATETIME IS NULL OR A_AD.END_DATETIME >= SYSDATE)
ORDER BY C_AD.SEQUENCE_NUMBER ASC)
where rownum = 1
If result set of above query is empty, then, it means we have finished one cycle of ADs scheduled for this channel by COPIA. Get the audio_ad_id with the lowest sequence number for the channel
SELECT * FROM (SELECT C_AD.AUDIO_AD_ID, A_AD.AD_LENGTH, A_AD.RA_FILE, A_AD.AA_SEARCH_STRING
FROM CHANNEL_AD C_AD, AUDIO_AD A_AD
WHERE C_AD.CHANNEL_ID = ?
AND C_AD.AUDIO_AD_ID = A_AD.AUDIO_AD_ID
AND A_AD.IS_ACTIVE > 0
AND (A_AD.START_DATETIME IS NULL OR A_AD.START_DATETIME <= SYSDATE)
AND (A_AD.END_DATETIME IS NULL OR A_AD.END_DATETIME >= SYSDATE)
ORDER BY C_AD.SEQUENCE_NUMBER ASC)
where rownum = 1
1504
is next audio_ad_id null?
NO
1505
YES
1506
NO
UPDATE CHANNEL SET AUDIO_AD_ID = ?, AUDIO_AD_LENGTH = ?, AUDIO_AD_RA_FILE = ?, SEARCH_STRING = ?, AUDIO_AD_SERVED = ? where CHANNEL_ID = ?
log WARN: no ads scheduled for this channel
increment AUDIO_AD TIMES_PROGRAMMED for the AUDIO_AD that was scheduled
1507
End of loop ?
1508
YES
sleep for 1 minute
1509

Figure 15

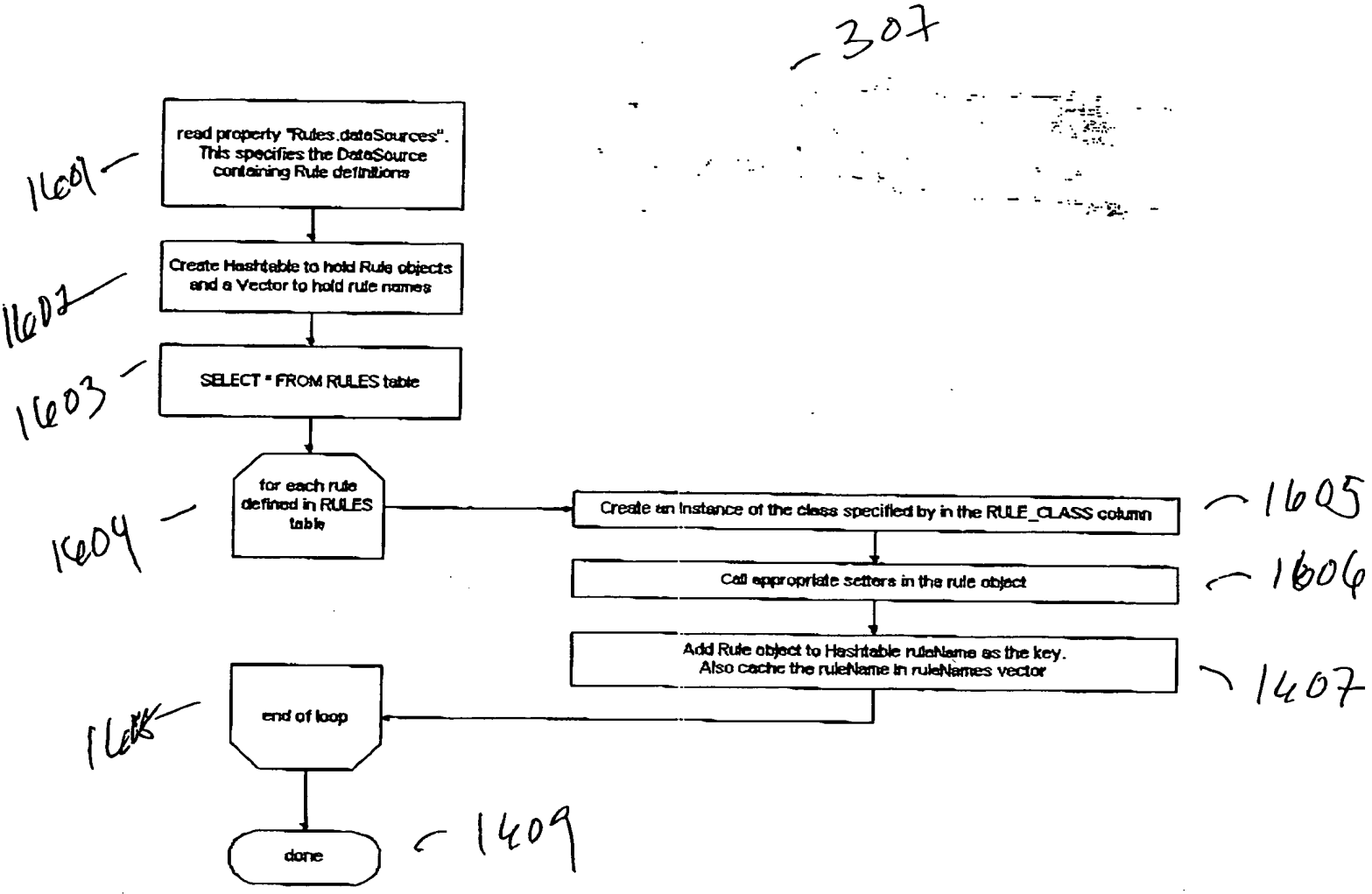
307
read property "Rules.dataSources". This specifies the DataSource containing Rule definitions
1601
Create Hashtable to hold Rule objects and a Vector to hold rule names
1602
SELECT * FROM RULES table
1603
for each rule defined in RULES table
1604
Create an instance of the class specified by in the RULE_CLASS column
1605
Call appropriate setters in the rule object
1606
Add Rule object to Hashtable ruleName as the key. Also cache the ruleName in ruleNames vector
1607
end of loop
1608
done
1609

Figure 16

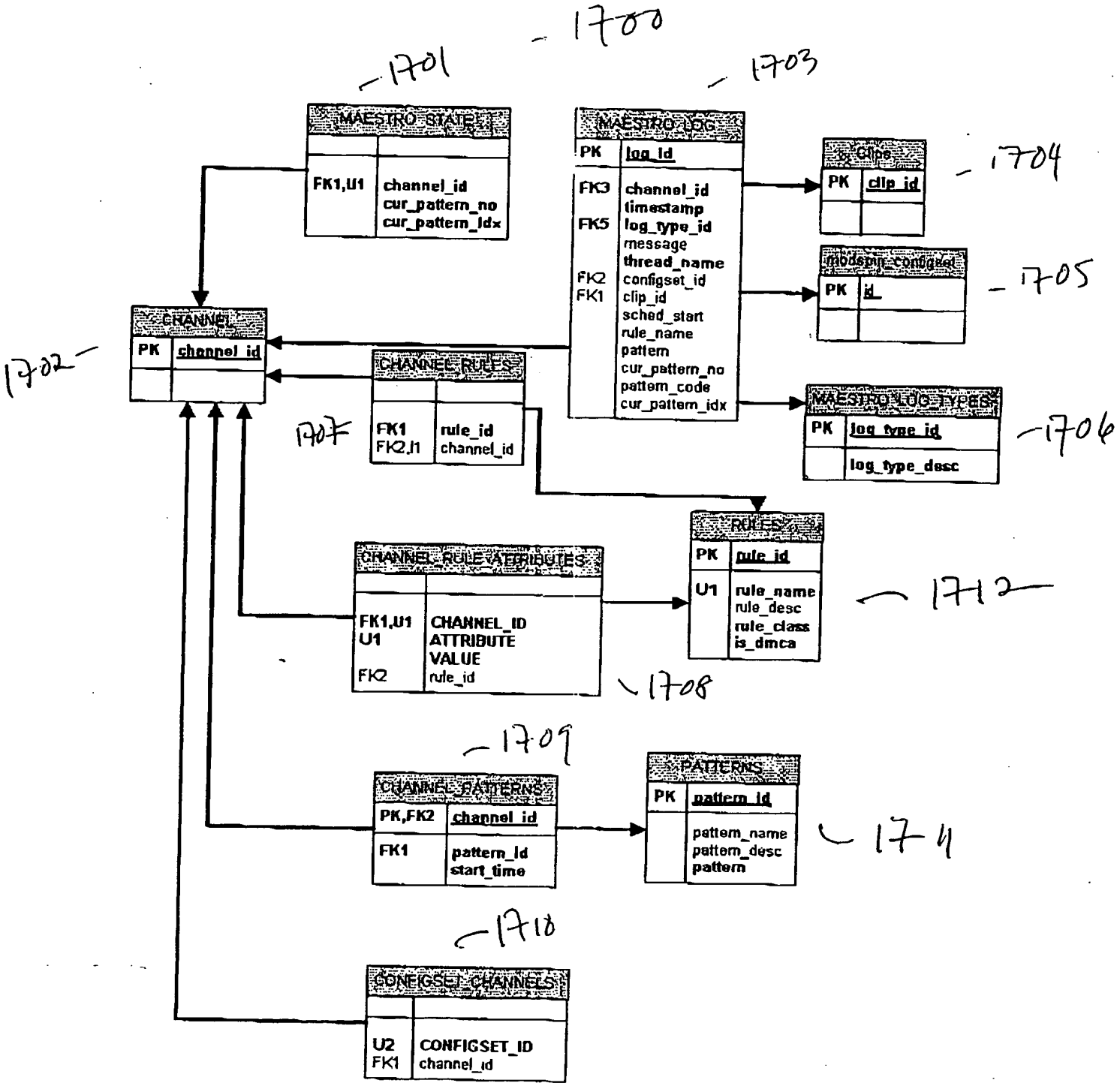
1700
1701
1703
MAESTRO_STATE
FK1,U1
channel_id
cur_pattern_no
cur_pattern_idx
MAESTRO_LOG
PK
log_id
FK3
channel_id
timestamp
FK5
log_type_id
message
thread_name
FK2
configset_id
FK1
clip_id
sched_start
rule_name
pattern
cur_pattern_no
pattern_code
cur_pattern_idx
Clips
PK
clip_id
1704
modestr_configset
PK
id
1705
1702
CHANNEL
PK
channel_id
CHANNEL_RULES
1707
FK1
FK2,I1
rule_id
channel_id
MAESTRO_LOG_TYPES
PK
log_type_id
log_type_desc
1706
RULES
PK
rule_id
U1
rule_name
rule_desc
rule_class
is_dmca
1712
CHANNEL_RULE_ATTRIBUTES
FK1,U1
U1
CHANNEL_ID
ATTRIBUTE
VALUE
FK2
rule_id
1708
1709
CHANNEL_PATTERNS
PK,FK2
channel_id
FK1
pattern_id
start_time
PATTERNS
PK
pattern_id
pattern_name
pattern_desc
pattern
1714
1710
CONFIGSET_CHANNELS
U2
FK1
CONFIGSET_ID
channel_id

Figure 17